When faced with NP-hard problems..

- A problem is NP-hard if it is at least as hard as an NP-complete problem.
  - This notion is not restricted to decision problems.
  - A problem does not need to be in NP to be NP-hard. (Recall, this is in contrast to an NP-complete problem.)

When faced with NP-hard problems..

- A problem is NP-hard if it is at least as hard as an NP-complete problem.
    - This notion is not restricted to decision problems.
    - A problem does not need to be in NP to be NP-hard. (Recall, this is in contrast to an NP-complete problem.)
- Many problems we face in practice are NP-hard. Only exponential-time algorithms are known for them.

When faced with NP-hard problems..

- A problem is NP-hard if it is at least as hard as an NP-complete problem.
  - This notion is not restricted to decision problems.
  - A problem does not need to be in NP to be NP-hard. (Recall, this is in contrast to an NP-complete problem.)
- Many problems we face in practice are NP-hard. Only exponential-time algorithms are known for them.
- What should we do?

When faced with NP-hard problems..

- A problem is NP-hard if it is at least as hard as an NP-complete problem.
  - This notion is not restricted to decision problems.
  - A problem does not need to be in NP to be NP-hard. (Recall, this is in contrast to an NP-complete problem.)
- Many problems we face in practice are NP-hard. Only exponential-time algorithms are known for them.
- What should we do?
  - Find relatively fast exponential-time algorithms. (Not all exponential-time algorithms are equally fast. The running time of an algorithm is exponential if it is bounded by $O(2^{n^c})$ for some $c > 0$.)

When faced with NP-hard problems..

- A problem is NP-hard if it is at least as hard as an NP-complete problem.
  - This notion is not restricted to decision problems.
  - A problem does not need to be in NP to be NP-hard. (Recall, this is in contrast to an NP-complete problem.)
- Many problems we face in practice are NP-hard. Only exponential-time algorithms are known for them.
- What should we do?
  - Find relatively fast exponential-time algorithms. (Not all exponential-time algorithms are equally fast. The running time of an algorithm is exponential if it is bounded by $O(2^{n^c})$ for some $c > 0$.)
  - Find polynomial-time algorithms for special cases of an NP-hard problem.

When faced with NP-hard problems..

- A problem is NP-hard if it is at least as hard as an NP-complete problem.
  - This notion is not restricted to decision problems.
  - A problem does not need to be in NP to be NP-hard. (Recall, this is in contrast to an NP-complete problem.)
- Many problems we face in practice are NP-hard. Only exponential-time algorithms are known for them.
- What should we do?
  - Find relatively fast exponential-time algorithms. (Not all exponential-time algorithms are equally fast. The running time of an algorithm is exponential if it is bounded by $O(2^{n^c})$ for some $c > 0$.)
  - Find polynomial-time algorithms for special cases of an NP-hard problem.
  - Design polynomial-time *approximation* algorithms.

Finding Small Vertex Covers

- Recall the vertex cover problem: given an undirected graph $G = (V, E)$ and an integer $k > 0$, is there a vertex cover of size at most k?

Finding Small Vertex Covers

- Recall the vertex cover problem: given an undirected graph $G = (V, E)$ and an integer $k > 0$, is there a vertex cover of size at most k?
- Requirement: if there is a vertex cover of size k, we would like to find it; otherwise we correctly report there is none.

Finding Small Vertex Covers

- Recall the vertex cover problem: given an undirected graph $G = (V, E)$ and an integer $k > 0$, is there a vertex cover of size at most k?
- Requirement: if there is a vertex cover of size k, we would like to find it; otherwise we correctly report there is none.
- Straightforward enumeration: takes time $\Omega(n^k)$, where $n = |V|$.

Finding Small Vertex Covers

- Recall the vertex cover problem: given an undirected graph $G = (V, E)$ and an integer $k > 0$, is there a vertex cover of size at most k?
- Requirement: if there is a vertex cover of size k, we would like to find it; otherwise we correctly report there is none.
- Straightforward enumeration: takes time $\Omega(n^k)$, where $n = |V|$.
- We would like an algorithm that runs in time $O(2^k \cdot \mathrm{poly}(n))$.

Finding Small Vertex Covers

- Recall the vertex cover problem: given an undirected graph $G = (V, E)$ and an integer $k > 0$, is there a vertex cover of size at most k?
- Requirement: if there is a vertex cover of size k, we would like to find it; otherwise we correctly report there is none.
- Straightforward enumeration: takes time $\Omega(n^k)$, where $n = |V|$.
- We would like an algorithm that runs in time $O(2^k \cdot \text{poly}(n))$.
- For reasonably large graph and small k, the difference is between impractical and practical.
  - Say $n = 1000$, $k = 10$.
  - $n^k \approx 2^{100}$.

Finding Small Vertex Covers

- Recall the vertex cover problem: given an undirected graph $G = (V, E)$ and an integer $k > 0$, is there a vertex cover of size at most k?
- Requirement: if there is a vertex cover of size k, we would like to find it; otherwise we correctly report there is none.
- Straightforward enumeration: takes time $\Omega(n^k)$, where $n = |V|$.
- We would like an algorithm that runs in time $O(2^k \cdot \text{poly}(n))$.
- For reasonably large graph and small k, the difference is between impractical and practical.
  - Say $n = 1000$, $k = 10$.
  - $n^k \approx 2^{100}$.
  - $2^k \cdot kn \approx 2^{24}$.

Designing the algorithm

Proposition

In a graph where no node has degree more than d and the number of edges is more than dk, then there is no vertex cover of size k.

Designing the algorithm

Proposition

In a graph where no node has degree more than d and the number of edges is more than dk, then there is no vertex cover of size k.

In particular, in a graph with n nodes and more than kn edges, there is no vertex cover of size k.

Designing the algorithm

Proposition

In a graph where no node has degree more than d and the number of edges is more than dk, then there is no vertex cover of size k.

In particular, in a graph with n nodes and more than kn edges, there is no vertex cover of size k.

Definition

For $u \in V$, $G - \{u\}$ is the graph obtained by deleting from G the node u and all edges incident to u.

Designing the algorithm

Proposition

In a graph where no node has degree more than d and the number of edges is more than dk, then there is no vertex cover of size k.

In particular, in a graph with n nodes and more than kn edges, there is no vertex cover of size k.

Definition

For $u \in V$, $G - \{u\}$ is the graph obtained by deleting from G the node u and all edges incident to u.

Proposition

Let $e = (u, v)$ be any edge of G. Then G has a vertex cover of size k if and only if at least one of $G - \{u\}$ and $G - \{v\}$ has a vertex cover of size $k - 1$.

Designing the algorithm

Proposition

In a graph where no node has degree more than d and the number of edges is more than dk, then there is no vertex cover of size k.

In particular, in a graph with n nodes and more than kn edges, there is no vertex cover of size k.

Definition

For $u \in V$, $G - \{u\}$ is the graph obtained by deleting from G the node u and all edges incident to u.

Proposition

Let $e = (u, v)$ be any edge of G. Then G has a vertex cover of size k if and only if at least one of $G - \{u\}$ and $G - \{v\}$ has a vertex cover of size $k - 1$.

This gives rise to a recursive algorithm.

Independent Sets on the Tree

- An optimization version of the Independent Set problem: given an undirected grpah $G = (V, E)$, find an independent set of the maximum cardinality.
- Recall that $S \subseteq V$ is an independent set if $\forall u, v \in S, (u, v) \notin E$.

Independent Sets on the Tree

- An optimization version of the Independent Set problem: given an undirected grpah $G = (V, E)$, find an independent set of the maximum cardinality.
- Recall that $S \subseteq V$ is an independent set if $\forall u, v \in S, (u, v) \notin E$.
- For general graphs the problem is NP-hard.

Independent Sets on the Tree

- An optimization version of the Independent Set problem: given an undirected grpah $G = (V, E)$, find an independent set of the maximum cardinality.
- Recall that $S \subseteq V$ is an independent set if $\forall u, v \in S, (u, v) \notin E$.
- For general graphs the problem is NP-hard.
- There is a polynomial-time algorithm for the problem if G is a tree.

Independent Sets on the Tree

- An optimization version of the Independent Set problem: given an undirected grpah $G = (V, E)$, find an independent set of the maximum cardinality.
- Recall that $S \subseteq V$ is an independent set if $\forall u, v \in S, (u, v) \notin E$.
- For general graphs the problem is NP-hard.
- There is a polynomial-time algorithm for the problem if G is a tree.

Proposition

For any leaf v in a tree, there is an independent set of maximum cardinality that contains v.

Independent Sets on the Tree

- An optimization version of the Independent Set problem: given an undirected grpah $G = (V, E)$, find an independent set of the maximum cardinality.
- Recall that $S \subseteq V$ is an independent set if $\forall u, v \in S, (u, v) \notin E$.
- For general graphs the problem is NP-hard.
- There is a polynomial-time algorithm for the problem if G is a tree.

Proposition

For any leaf v in a tree, there is an independent set of maximum cardinality that contains v.

Proposition

For any maximum independent set S and $v \in S$, $S - \{v\}$ is a maximum independent set for the graph formed from G by removing v and all its neighbors.

Independent Sets on the Tree

- An optimization version of the Independent Set problem: given an undirected grpah $G = (V, E)$, find an independent set of the maximum cardinality.
- Recall that $S \subseteq V$ is an independent set if $\forall u, v \in S, (u, v) \notin E$.
- For general graphs the problem is NP-hard.
- There is a polynomial-time algorithm for the problem if G is a tree.

Proposition

For any leaf v in a tree, there is an independent set of maximum cardinality that contains v.

Proposition

For any maximum independent set S and $v \in S$, $S - \{v\}$ is a maximum independent set for the graph formed from G by removing v and all its neighbors.

The two propositions gives rise to a greedy algorithm.

Weighted Independent Set on the Tree

- Problem: Given an undirected graph $G = (V, E)$ with nonnegative weights on the vertices: $w : V \to \mathbb{R}_+$. Find an independent set S with maximum total weight: $\sum_{v \in S} w(v)$.

Weighted Independent Set on the Tree

- Problem: Given an undirected graph $G = (V, E)$ with nonnegative weights on the vertices: $w : V \to \mathbb{R}_+$. Find an independent set S with maximum total weight: $\sum_{v \in S} w(v)$.
- The problem is at least as hard as the unweighted version, and so is NP-hard.

Weighted Independent Set on the Tree

- Problem: Given an undirected graph $G = (V, E)$ with nonnegative weights on the vertices: $w : V \to \mathbb{R}_+$. Find an independent set S with maximum total weight: $\sum_{v \in S} w(v)$.
- The problem is at least as hard as the unweighted version, and so is NP-hard.
- When G is a tree, the problem can be solved in polynomial time by a dynamic programming algorithm.

Weighted Independent Set on the Tree

- Problem: Given an undirected graph $G = (V, E)$ with nonnegative weights on the vertices: $w : V \to \mathbb{R}_+$. Find an independent set S with maximum total weight: $\sum_{v \in S} w(v)$.
- The problem is at least as hard as the unweighted version, and so is NP-hard.
- When G is a tree, the problem can be solved in polynomial time by a dynamic programming algorithm.
- Let children(v) denote the set of children of node v.

Weighted Independent Set on the Tree

- Problem: Given an undirected graph $G = (V, E)$ with nonnegative weights on the vertices: $w : V \to \mathbb{R}_+$. Find an independent set S with maximum total weight: $\sum_{v \in S} w(v)$.
- The problem is at least as hard as the unweighted version, and so is NP-hard.
- When G is a tree, the problem can be solved in polynomial time by a dynamic programming algorithm.
- Let children(v) denote the set of children of node v.
- Key observation: in considering whether to choose the root v:

Weighted Independent Set on the Tree

- Problem: Given an undirected graph $G = (V, E)$ with nonnegative weights on the vertices: $w : V \to \mathbb{R}_+$. Find an independent set S with maximum total weight: $\sum_{v \in S} w(v)$.
- The problem is at least as hard as the unweighted version, and so is NP-hard.
- When G is a tree, the problem can be solved in polynomial time by a dynamic programming algorithm.
- Let children(v) denote the set of children of node v.
- Key observation: in considering whether to choose the root v:
  - if we do not select v, we should take the union of the optimal solutions for the subtrees rooted at children(v);

Weighted Independent Set on the Tree

- Problem: Given an undirected graph $G = (V, E)$ with nonnegative weights on the vertices: $w : V \to \mathbb{R}_+$. Find an independent set S with maximum total weight: $\sum_{v \in S} w(v)$.
- The problem is at least as hard as the unweighted version, and so is NP-hard.
- When G is a tree, the problem can be solved in polynomial time by a dynamic programming algorithm.
- Let children(v) denote the set of children of node v.
- Key observation: in considering whether to choose the root v:
  - if we do not select v, we should take the union of the optimal solutions for the subtrees rooted at children(v);
  - if we select v, we should take the union of the optimal solutions for the subtrees rooted at children(v) but subject to that children(v) are not selected.

Weighted Independent Set on the Tree

- Problem: Given an undirected graph $G = (V, E)$ with nonnegative weights on the vertices: $w : V \to \mathbb{R}_+$. Find an independent set S with maximum total weight: $\sum_{v \in S} w(v)$.
- The problem is at least as hard as the unweighted version, and so is NP-hard.
- When G is a tree, the problem can be solved in polynomial time by a dynamic programming algorithm.
- Let children(v) denote the set of children of node v.
- Key observation: in considering whether to choose the root v:
  - if we do not select v, we should take the union of the optimal solutions for the subtrees rooted at children(v);
  - if we select v, we should take the union of the optimal solutions for the subtrees rooted at children(v) but subject to that children(v) are not selected.
  - But "optimal solution for a tree subject to that its root is not selected" is precisely the first case.